IN THE SPECIFICATION:

Please replace the paragraph on page 3 between lines 7 and 15 with the following amended paragraph:

Unfortunately, in the "incomplete knowledge" case, there are several additional problems that can arise. For example, when the runtime environment of the program to be compiled will consist of several separately compiled pieces or modules, it may be discovered that the same subroutine (or other software component) was included in more than one separately compiled piece or module. This would not be a problem for purely procedural subroutines, but when the subroutines have static storage of one form or another (including the static data structures associated with C++ or ~~Java®~~JAVA® programming language classes) then errors will result unless a single static storage image is somehow shared between all copies.

Please replace the paragraph on page 4 between lines 11 and 20 with the following amended paragraph:

This invention addresses these and other problems associated with the prior art by providing a computer system, a computer product, a method and a framework in which static storage within an environment comprising a plurality of compilation modules is managed such that compiled cloned copies of called externally resolved (with respect to a compilation unit) items are preferentially executed in favor of the corresponding externally resolved item based on a favorable comparison of version information prior to execution. The cloned copies are compiled in a manner providing internal resolution (with respect to the compilation unit) by, for example, in-line coding. In one embodiment, ~~Java®~~JAVA® programming language methods are processed within the context of a modified framework.

On page 5, please replace the **BRIEF DESCRIPTION OF THE DRAWINGS** (excepting the header itself) with the following amended paragraph:

The teachings of the present invention can b readily understood by considering the following detailed description in conjunction with the accompanying drawing in which:

FIGURE 1 is a block diagram of a computer system consistent with the invention;

FIGURE 2 is a block diagram of an exemplary software environment for the computer system of FIGURE 1;

FIGURE 3A depicts a code segment illustrating a direct call of a subroutine;

FIGURE 3B depicts a code segment illustrating an indirect call of a subroutine;

FIGURE 4 depicts a code segment 400 illustrating an inlined access of a subroutine;

FIGURE 5 depicts a code segment illustrating an optimization enabled by the inlining technique of FIGURE 4;

FIGURE 6 depicts a code segment illustrating the use of multiple copies of statically stored objects;

FIGURE 7 depicts a code segment useful in understanding the present invention, and more particularly illustrates a problem that can occur with static storage if multiple copies of a class attempt to share the same static;

FIGURE 8A depicts a code segment and address space utilization useful in understanding the present invention;

FIGURE 8B depicts a reentrant static addressing scheme utilizing the code segment of FIGURE 8A;

FIGURE 9 depicts a data structure representing a ~~Java®~~JAVA® programming language class file;

FIGURE 10 depicts a data structure representing a loaded ~~Java®~~JAVA® programming language class;

FIGURE 11 depicts a data structure representing a compiled ~~Java®~~JAVA® programming language class;

FIGURES 12 and 13 depict code segments useful in understanding the invention;

FIGURE 14 depicts a graphical representation useful in understanding the invention;

FIGURE 15 depicts a plurality of data structures illustrating constant pool entries and constant resolution ntries;

FIGURES 16A and 16B (collectively referred to as FIGURE 16) depict a flow diagram of a constant resolution process;

FIGURE 17 depicts a graphical representation useful in understanding the invention;

FIGURE 18 depicts a flow diagram of a process according to the invention; and

FIGURE 19 depicts a flow diagram of a method for compiling an externally resolved subroutine according to an embodiment of the invention.

To facilitate understanding, identical reference numerals have been used, where possible, to designate identical elements that are common to the figures.

Please replace the paragraph on page 10 between lines 22 and 29 with the following amended paragraph:

The present invention functions by dividing categories (or subcategories of the categories) of static items storage into two super-categories: Those categories that are unique to an individual compiled instance of a subroutine, and those categories that are shareable between individual compiled instances, provided that they all reflect the same source version. Note that there may be some items that may be placed into either super-category, as they are not malleable (and hence don't need to be shares) but also do not contain information unique to the particular compiled instance of a subroutine. Such things as the name of the ~~Java®~~JAVA® programming language class would fall into this group.

Please replace the paragraph on page 11 between lines 4 and 12 with the following amended paragraph:

When a compiler determines that a subroutine (e.g., a ~~Java®~~JAVA® programming language method or other procedure) should be copied into a compilation unit where it was not already defined, a "clone" copy of the permanent data structures for the containing class is also made. These data structures include primarily the main class structure, the tables of static and instance fields, the table of methods, and the

constant pool. (The "virtual method table" does not need to be defined at this time so long as the rules for constructing it are known. Also, the field tables do not need to be unique per method instance and hence do not need to be copied so long as addressability to the originals can be maintained.)

Please replace the paragraph on page 13 between line 20 and page 14 line 3 with the following amended paragraph:

~~Java®~~ JAVA® programming language Implementation

An embodiment of the invention as applied to the ~~Java®~~JAVA® programming language environment will now be described in more detail. In the case of a typical ~~Java®~~JAVA® programming language implementation, the information that resides in the static storage accessible to a given subroutine can be roughly categorized as follows:

(a) The main ~~Java®~~JAVA® programming language class structure, containing such information as the name of the class and its authorization attributes; (b) A table of static fields defined by the class; (c) A table of instance fields defined by the class; (d) The actual static data storage areas; (e) A table of methods actually implemented by this class; (f) A "virtual method table" identifying all virtual methods either implemented or inherited by this class; (g) A constant pool table, containing malleable entries used to address classes, methods, fields, and strings that cannot be absolutely determined at compile time; (h) Constant resolution entries, in lists associated with each method (i.e., each procedure or sub-routine) implemented (not inherited) by the class; and (i) Static storage created by the actual code generation process, including, for example, storage for numeric literals that are too large to be placed into a register using an "immediate form" machine instruction.

Please replace the paragraph on page 14 between lines 4 and 12 with the following amended paragraph:

FIGURE 9 depicts a data structure representing a ~~Java®~~JAVA® programming language class file 900 including at least a class description field 910, static field definitions 920, instance field definitions 930, method definitions 940 and a constant pool 950. The file contains a general class description, static and instance field

definitions (which in practice are merged together with flag bits indicating which type is which), method d finitions, and a constant pool. The constant pool centralizes most of the literal string values and external references used within the class, both allowing a more compact representation (due to shared used of the literal values) and providing a convenient prototype for the static storage to be used by the class during execution.

Please replace the paragraph on page 14 between lines 13 and 26 with the following amended paragraph:

FIGURE 10 depicts a data structure representing a loaded ~~Java®~~JAVA® programming language class 1000 including the contents (910 - 950) of the ~~Java®~~JAVA® programming language class file 900 of FIGURE 9, though augmented with additional information such as addresses, offset, and authorities that can be determined once a class is loaded. In addition, the actual static storage (1060) for static items specified in the ~~Java®~~JAVA® programming language source is allocated, along with a virtual method table (1070). The location in the static storage are indirectly addressed by pointers from the augmented static field definitions (or, in some designs, the static storage for fields can actually be allocated within the augmented static field definition entries). The virtual method table contains a list of all methods belonging to this class, including inherited ones. The order of entries in this table is generally arbitrary, except that inherited methods appear first, and are in the same order as they appear in the class from which they were inherited. This allows the efficient addressing of virtual methods once the name of a method is resolved to an index into the table (an operation that needs to be performed at most once for each call site).

Please replace the paragraph on page 14 between lines 27 and 33 with the following amended paragraph:

In many cases, (such as the ~~Java®~~JAVA® programming language implementation on the AS/400 computer system manufactured by International Business Machines Corporation of Armonk, New York), the constant pool is addressed indirectly via an ordered table of pointers. That is, a constant pool vector table contains of an ordered array of pointers, with each pointer addressing the corresponding pool

entry. This simplifies the addressing of constant pool entries (as they are not of uniform size), at the expense of an extra level of indirection.

Please replace the paragraph on page 15 between lines 1 and 8 with the following amended paragraph:

FIGURE 11 depicts a data structure representing a compiled ~~Java®~~JAVA® programming language data class 1100 such as implemented on the AS/400 computer system. The compiled class structure 1100 comprises the five items (910 - 950) of the ~~Java®~~JAVA® programming language class file 900 of FIGURE 9, though augmented by additional information that could be derived during the compilation process. In addition, the compiled ~~Java®~~JAVA® programming language class structure 1100 of FIGURE 11 comprises four additional items, namely, verification directives 1110, constant resolution directives 1120, code generation static storage 1130 and the compiled code 1140.

Please replace the paragraph on page 15 between lines 20 and 22 with the following amended paragraph:

The code 1140 is the representation, in machine instructions, of the methods within the ~~Java®~~JAVA® programming language class. Addressability to the entry points of the methods within this item is recorded in the augmented method table entries.

Please replace the paragraph on page 15 between lines 23 and 29 with the following amended paragraph:

Also note that, during the compilation process, additional constant pool items may be added to the constant pool. For instance, since the operand passed to an ATHROW ~~Java®~~JAVA® programming language operation must be a subclass of Throwable, it may be necessary to add a constant for Throwable to the constant pool so that it can be referenced by the verification directives. Also, in the case of references to "cloned" classes as described in this invention, it is usually necessary to add additional class and method constants to the constant pool to represent the "cloned" methods and classes".

Please replace the paragraph on page 17 between lines 23 and 26 with the following amended paragraph:

An object 1420 contains a pointer of the basic class description 1421 of the class of which the object is an instance, some object status information 1422 (e.g., lock state), and the actual object data 1423 corresponding to the object fields declared in the ~~Java®~~JAVA® programming language source.

Please replace the paragraph on page 20 between lines 29 and 32 with the following amended paragraph:

FIGURE 17 depicts a graphical representation useful in understanding a ~~Java®~~JAVA® programming language realization of the present invention. Specifically, FIGURE 17 depicts a clone basic class description 1710, a parent basic class description 1720, a clone full class description 1730 and a parent full class description 1740.

Please replace the paragraph on page 21 between lines 1 and 10 with the following amended paragraph:

The clone basic class description 1710 includes at least a clone constant pool 1711, a MethodRef 1712, a FieldRef 1713, a clone basic class description 1714 and a clone virtual method table 1715. The parent basic class description includes at least a parent constant pool 1721, a parent method reference 1722, a parent field reference 1723, a parent basic class description 1724 and a parent virtual method table 1725. The clone full class description 1730 includes at least a clone full class description 1731, a clone method table 1732. The parent full class description 1740 includes at least a parent full class description 1741, parent method table 1742 and a ~~Java®~~JAVA® programming language static storage 1743. It will be appreciated by those skilled in the art that only portions of the various data structures are depicted.

Please replace the paragraph on page 21 between lines 23 and 29 with the following amended paragraph:

Th clone basic class description 1714 includes a first linkage 1714A to the clon full class description 1730, and a second linkage 1714B to the par nt basic class description 1724. The parent basic class description 1724 includes an entry 1724A having a linkage to the parent full class description 1740. The field references 1713 and 1723 of, respectively, the clone basic class description 1710 and the parent basic class description 1720 have linkages to the ~~Java®~~JAVA® programming language static storage 1743. The method reference 1722 has a linkage to the parent method table 1742.

Please replace the paragraph on page 22 between lines 6 and 27 with the following amended paragraph:

Prior to execution of the compiled code, the class for the method that will be initially referenced must be "loaded." This operation consists of finding the class data structures corresponding to that class and making them ready. This basic process is described in greater detail in "The ~~Java®~~JAVA® programming language Virtual Machine Specification – Second Edition" (ISBN 0-201043294-3). The basic process is modified herein in that rather than loading the class data structures directly from a class file (and interpreting them anew), the pre-processed permanent class data structures associated with the compiled code are used. This concept is described in more detail in commonly assigned U.S. patent application No. 09/024,111, filed February 17, 1998 and incorporated herein by reference in its entirety. Briefly, native program code is associated with an executable file containing platform-independent code, e.g., a ~~Java®~~JAVA® programming language class file. Given that program code associated with a file attribute is typically transparent to conventional ~~Java®~~JAVA® programming language interpreters, the performance of a ~~Java®~~JAVA® programming language or other platform-independent computer program is enhanced for operation on a particular platform while maintaining platform-independence. That is, by associated alternate program code with an executable file using a file attribute, the alternate program code is more transparent to the user, as well as to conventional execution modules that are not specifically configured to detect and execute the alternate program code. Moreover, system write access is often not required for a user, so that the integrity of the original

executable program code can be protect d. Thus, alternate program code is associated with an executable file using a file attribute so that the alternate program code may be retrieved and executed in appropriate circumstances.

Please replace the paragraph on page 22 between line 28 and page 23, line 14 with the following amended paragraph:

After the class is loaded, and before the instructions of the initial method can be executed, resolution must be performed for any tightly-bound references called out in the initial method. The required resolution operations are identified via the constant resolution entries described earlier. During the resolution operation, several processes are performed as follows. First, the constant pool item which references the tightly-bound references is resolved. Second, for string constants, the string is constructed and "interned" and the address of the string is made available to the compiled code. Third, for instance fields whose offset has been tightly bound into the compiled code, the expected offset is compared to the offset as determined by the class loader, and an error is detected if these do not match. Fourth, for methods whose entry points have been tightly bound into the compiled code, the structure within the compilation unit that identifies each such method and describes its characteristics (i.e., the "permanent method table entry") is located and several operations are performed. First, the version of the class to which the resolved-to method belongs is compared to the version of the class that was copied during the compilation process. If the versions do not match then an error is detected. Second, addressability to the static storage of the target ~~Java®~~JAVA® programming language method is established so that it can be passed as a hidden parameter when the target ~~Java®~~JAVA® programming language method is called. Third, the first and second resolution operations are recursively performed for tightly-bound references from the target- ~~Java®~~JAVA® programming language method.

Please replace the paragraph on page 23 between lines 17 and 26 with the following amended paragraph:

The above-described resolution operation can be triggered several ways. The most convenient approach is to replace the pointer to the entry point of the method

with a pointer to a resolution routine. After the resolution routine has been executed once, then the correct pointer to the method entry point is restored. Using this technique, the above resolution operations are recursively performed for tightly-bound references from the target ~~Java®~~JAVA® programming language method. That is, since tightly-bound calls do not operate indirectly through the ~~Java®~~JAVA® programming language method pointer, the corresponding initialization of the target method will not occur unless done upon entry to the calling method. Implementations using different techniques to trigger the resolution process do not necessarily require the recursive resolution operation to be performed.

Please replace the paragraph on page 23 between line 27 and page 24, line 2 with the following amended paragraph:

When the ~~Java®~~JAVA® programming language method begins execution, it is passed, via a hidden parameter, addressability to a runtime version of its entry in the containing class's method table. This serves as the "basic static storage" of the method. The runtime version of the method table entry contains, directly or indirectly, addressability to all other static data items. In particular, it contains addressability to a subset of the runtime class data structure fields, comprising a fixed-length basic data structure, a prepended constant pool vector table for the class and a postpended virtual method vector table for the class.

Please replace the paragraph on page 24 between lines 3 and 13 with the following amended paragraph:

The fixed-length basic data structure. In the current implementation these fields represent a direct copy of the first portion of the full runtime class data structure with minor modifications. While not necessary to the invention, this allows the full and subset structures to be used interchangeably in some circumstances. The subset structure contains, among other things, a pointer to itself, a pointer to the full structure, a pointer to the ~~Java®~~JAVA® programming language object of class java.lang.Class that represents the class in ~~Java®~~JAVA® programming language code, and the offset of the first instance field (beyond superclass fields) in an object of this class. Together

with the following two items this structure provides all of the addressability that may be required by the executing ~~Java®~~JAVA® programming language code, and a single pointer to the start of this fixed-length structure serves to address all three items.

Please replace the paragraph on page 25 between lines 25 and 33 with the following amended paragraph:

When, during the processing of constant resolution entries for a method (prior to first entry to that method), an entry is encountered which is marked to indicate it is a reference to a copied ("cloned") method, the class for the reference is initially resolved in a normal fashion. That is to say, the ~~Java®~~JAVA® programming language class loader mechanism is invoked to locate the class in the "classpath" associated with the execution environment. If no such class is found then an error reported in the normal fashion. (Note that such an error may in some cases be recorded and then deferred in order to assure conformant operation of the ~~Java®~~JAVA® programming language execution environment.) If such a class is found then it is loaded via the usual mechanism as described earlier.

Please replace the paragraph on page 28 between lines 20 and 25 with the following amended paragraph:

A ~~Java®~~JAVA® programming language method containing tightly bound references and all the methods that it is tightly bound to, and all the methods that those methods are tightly bound to, etc, comprise a directed graph (that may contain cycles). Whenever a method is entered for the first time via an indirect branch, the graph so defined for that method is walked (taking care to avoid infinite recursion in the case of cycles) and the appropriate "resolution operations" are applied to all the methods in the graph.

On page 35, please replace the **ABSTRACT OF THE DISCLOSURE** (excepting the header itself) with the following amended paragraph:

A computer system, a computer product and a method in which static storage within an nvironment comprising a plurality of compilation modules is managed such

that compiled cloned copies of called externally resolved (with respect to a compilation unit) items are preferentially executed in favor of the corresponding xternally resolv d item based on a favorable comparison of version information prior to execution. In one embodiment, ~~Java®~~JAVA® programming language methods are processed within the context of a modified framework.